

Database Performance at Gitlab.com

Jose Cores Finotto,
GitLab,
Staff Database Reliability Engineer



Nikolay Samokhvalov,
Postgres.ai,
Founder







Database Performance at Gitlab.com

Jose Cores Finotto,
GitLab,
Staff Database Reliability Engineer



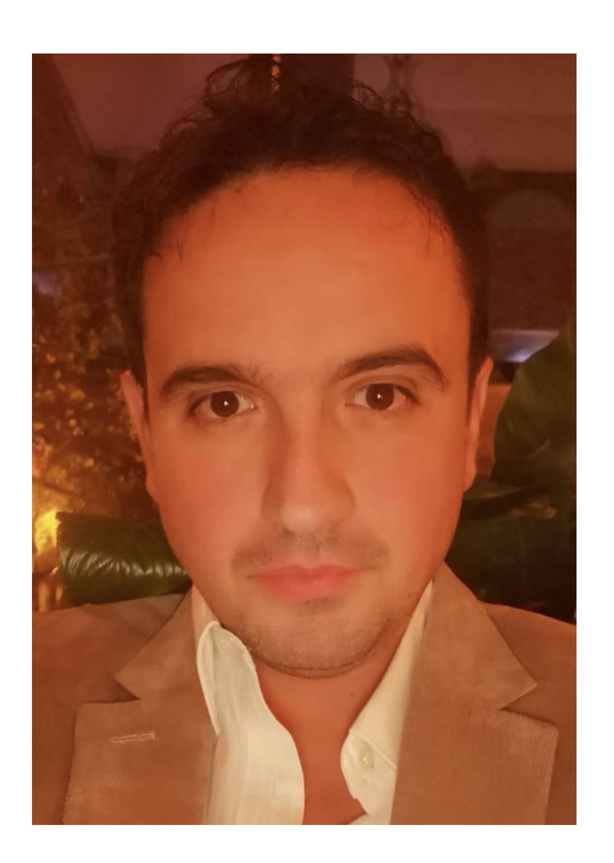
Nikolay Samokhvalov,
Postgres.ai,
Founder

Speaker: Jose Cores Finotto

- My name is Jose Cores Finotto I work with the Infrastructure team at GitLab.

- I have been a part of the GitLab team since September 2018.

- Background in large organizations with extensive experience in Infrastructure, especially in relational databases.



Speaker: Nikolay Samokhvalov



- Database systems:
  - 2002-2005:



  - since 2005:





- Worked on XML data type and functions (2005-2007)
- Long-term community activist – #RuPostgres, Postgres.tv



- Conferences Program Committee







etc.

- Current business:



Agenda

- GitLab
- Architecture and challenges
- Performance analysis
- postgres-checkup
- Joe Bot & Database Lab

Gitlab Values

1

Collaboration

Work asynchronously with fully remote workforce (org)

Use GitLab to build GitLab, there's an Issue and/or Merge Request for everything

2

Results

Track outcomes, not hours

3

Efficiency

Straightforward solutions win. Complexity slows cycle time.

4

Diversity

Remote-only tends toward global diversity, but we still have a ways to go.

Hire those who add to culture, not those who fit with it. We want **cultural diversity** instead of cultural conformity.

5

Iteration

Minimum Viable Change (MVC) if the change is better than the existing solution, ship it.

6

Transparency

Everything at GitLab is **public by default**: Strategy, Roadmap, Quarterly Goals, Handbook, and Issue Trackers

The open source project



Used by more than

100,000

organizations

A community of

3,000+

code contributors



We release every month on the 22nd and there is a publicly viewable direction for the product.

Learn more from our blog →

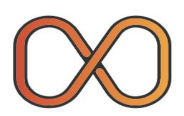
The company

The company



GitLab Inc. is an open-core company that sells subscriptions that offer more features and support for GitLab.

Learn about open core →



GitLab, the product is a complete DevOps platform, delivered as a single application, fundamentally changing the way Development, Security, and Ops teams collaborate.

Learn more about our product →

All remote with
1297
team members

Over
30 million
estimated registered users

Located in
67
countries

The Company

2011

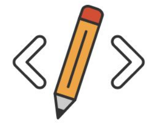
GitLab, the open source project began.

2015

We joined Y Combinator and started growing faster.
Join our team.



Most of our internal procedures can be found in a publicly viewable 5000+ page handbook and our objectives are documented in our OKRs.



Our mission is to change all creative work from read-only to read-write so that **everyone can contribute.** This is part of our overall strategy.



Our values are Collaboration, Results, Efficiency, Diversity, Inclusion & Belonging , Iteration, and Transparency (CREDIT) and these form an important part of our culture.



Our Tanuki (Japanese for raccoon dog) logo symbolizes our values with a smart animal that works in a group to achieve a common goal, you can download it on our press page.

Features





Product Pricing Resources Blog Support Install GitLab Explore Sign in **Get free trial**

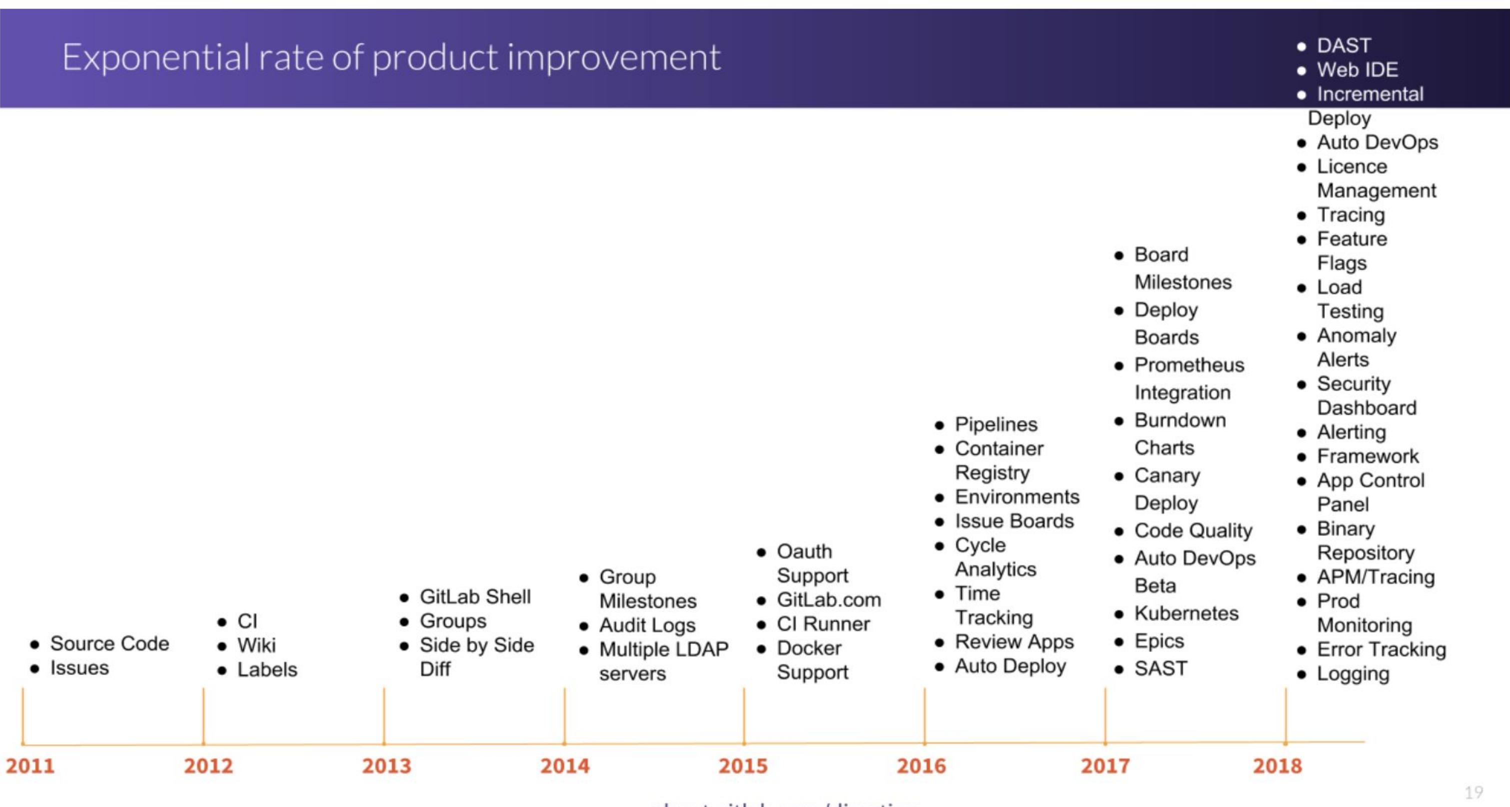
The definitive guide to remote work Download the playbook

Discover a more streamlined way to work

GitLab is a complete DevOps platform, delivered as a single application.

Manage	Plan	Create	Verify	Package	Secure	Release	Configure	Monitor	Protect
Subgroups	Issue Tracking	Source Code Management	Continuous Integration (CI)	Package Registry	SAST	Continuous Delivery	Auto DevOps	Runbooks	Container Scanning
Audit Events	Time Tracking	Code Review	Code Quality	Container Registry	DAST	Pages	Kubernetes Management	Metrics	Container Host Security
Audit Reports	Boards	Wiki	Code Testing and Coverage	Helm Chart Registry	Fuzz Testing	Review Apps	ChatOps	Incident Management	Container Network Security
Compliance Management	Epics	Static Site Editor	Load Testing	Dependency Proxy	Dependency Scanning	Advanced Deployments	Serverless	Logging	
Code Analytics	Roadmaps	Web IDE	Web Performance	Release Evidence	License Compliance	Feature Flags	Infrastructure as Code	Tracing	
DevOps Reports	Service Desk	Live Preview	Usability Testing	Git LFS	Secret Detection	Release Orchestration	Cluster Cost Management	Error Tracking	
Value Stream Management	Requirements Management	Snippets	Accessibility Testing		Vulnerability Management	Secrets Management		Product Analytics	
Insights	Quality Management		Merge Trains						
	Design Management								

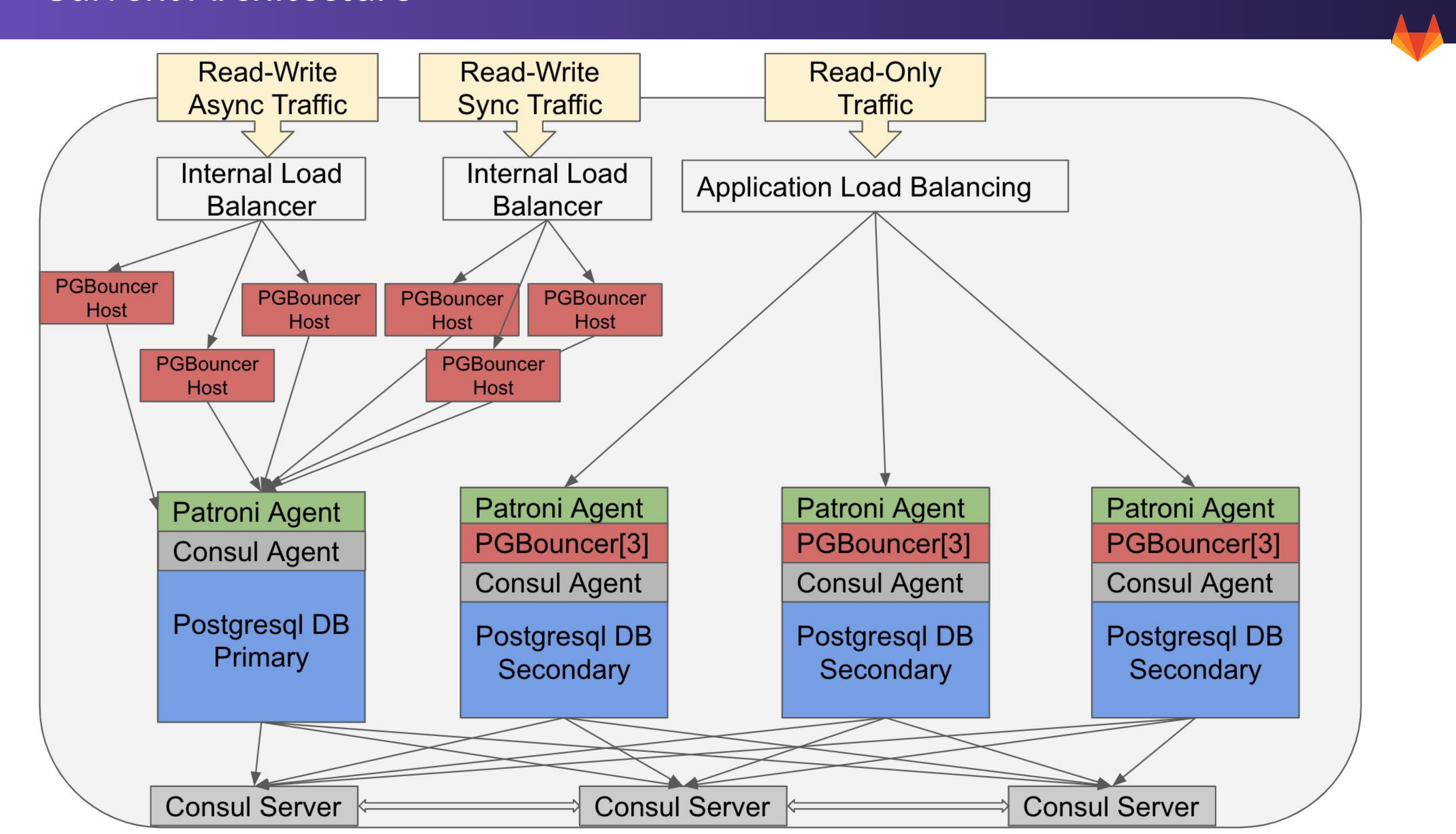
Feature development matrix



We have a hosted version of Gitlab:

- Over 40 million daily git pull operations.
- More than 6k git requests per second.
- 750.000 git pushes a day.
- 60k to 80k transactions per second on the database
- 8 database replicas and 1 primary
- Database size : 14 TiB
- Hardware architecture GCP 96 cores with 624 GiB of RAM.

Current Architecture



Performance degradation analysis

Database performance peak - 12 of January - 16:06 AM



The following CPU utilization peak started at 16:05, reaching 87%:

environment gprd Host patroni-06-db-gprd.c.gitlab-production.internal

CPU usage

% of all CPUs

90.0% 80.0% 70.0% 60.0% 50.0% 40.0% 30.0% 20.0% 10.0% 0%

15:58 15:59 16:00 16:01 16:02 16:03 16:04 16:05 16:06 16:07 16:08 16:09 16:10 16:11 16:12

iowait irq nice softirq steal system user

- [x] Evaluate the analysis report, metrics and queries. If applies, create new issues with the label `infradev` or `datastores` to propose new improvements to the database cluster overall.

Performance degradation analysis



Jose Finotto @Finotto · 1 day ago







We had the following top 10 statements **by total time** in execution during this peak:

Query:

```
topk(10,
  sum by (queryid) (
    rate(pg_stat_statements_seconds_total{env="gprd", monitor="db", type="patroni",instance="patroni-06-db-gp
  )
)
```

In this analysis, we are considering a 15 minutes interval.

https://thanos-query.ops.gitlab.net/graph?g0.range_input=15m&g0.end_input=2021-01-12%2016%3A15&g0.step_input=10&g0.max_source_resolution=0s&g0.expr=topk(10%2C%20%0A%20%20sum%20by%20(queryid)%20(%0A%20%20%20%20rate(pg_stat_statements_seconds_total%7Benv%3D%22gprd%22%2C%20monitor%3D%22db%22%2C%20type%3D%22patroni%22%2Cinstance%3D%22patroni-06-db-gprd.c.gitlab-production.internal%3A9187%22%7D%5B1m%5D)%0A%20%20)%0A)&g0.tab=0

Performance degradation analysis



Jose Finotto @Finotto · 1 day ago

Owner

The outputs are:

Enable query history

```
topk(10,
  sum by (queryid) (
    rate(pg_stat_statements_seconds_total{env="gprd", monitor="db", type="patroni",instance="patroni-06-db-gprd.c.gitlab-production.internal:9187"}[1m])
  )
)
```

Load time: 289m
Resolution: 10s
Total time series

Execute | - insert metric at cursor - | deduplication | partial response

Graph | Console

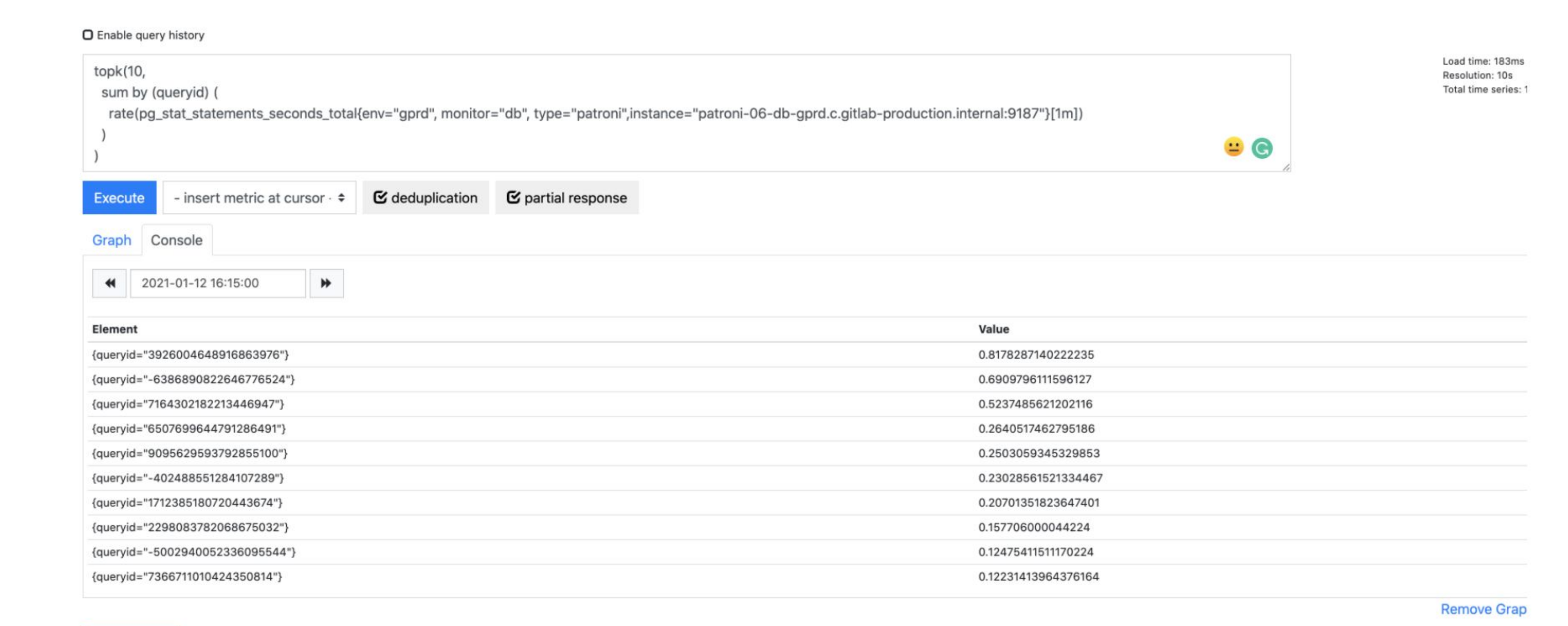
− 15m + | 2021-01-12 16:15 | 10 | stacked | Only raw data

8
6
4
2
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

[redacted]

Performance degradation analysis



Performance degradation analysis



Jose Finotto @Finotto · 1 day ago

Owner  

Those queryIds are the following SQL statements:

QueryId	Query
3926004648916863976	SELECT "ci_builds".* FROM "ci_builds" INNER JOIN "projects" ON "projects"."id" = "ci_builds"."project ci_builds.project_id = project_features.project_id LEFT JOIN (SELECT "ci_builds"."project_id", count() *"ci_builds"."type" = $1 AND ("ci_builds"."status" IN ($2)) AND "ci_builds"."runner_id" IN (SELECT "ci_r "ci_runners"."runner_type" = $3) GROUP BY "ci_builds"."project_id") AS project_builds ON ci_builds.p ("ci_builds"."status" IN ($4)) AND "ci_builds"."runner_id" IS NULL AND "projects"."shared_runners_en = $6 AND (project_features.builds_access_level IS NULL or project_features.builds_access_level > $7 ("projects"."visibility_level" = $9 OR (EXISTS (WITH RECURSIVE "base_and_ancestors" AS ((SELECT '*(namespaces.id = projects.namespace_id)) UNION (SELECT "namespaces".* FROM "namespaces", "b "namespaces"."id" = "base_and_ancestors"."parent_id")) SELECT $10 FROM "base_and_ancestors" A namespace_statistics ON namespace_statistics.namespace_id = namespaces.id WHERE "namespace (COALESCE(namespaces.shared_runners_minutes_limit, $11, $12) = $13 OR COALESCE(namespace_ COALESCE((namespaces.shared_runners_minutes_limit + COALESCE(namespaces.extra_shared_run COALESCE(namespaces.extra_shared_runners_minutes_limit, $17)), $18) * $19)))) AND (NOT EXISTS "taggings"."taggable_type" = $21 AND "taggings"."context" = $22 AND (taggable_id = ci_builds.id) AI ORDER BY COALESCE(project_builds.running_builds, $25) ASC, ci_builds.id ASC */application:web,correlation_id:01EVX3GF3VGAVE6TYFMR82EJFN/*
-6386890822646776524	SELECT "users".* FROM "users" INNER JOIN "project_authorizations" ON "users"."id" = "project_auth "project_authorizations"."project_id" = $1 */application:web,correlation_id:Lmz5Aaf8Vpa/*
7164302182213446947	UPDATE "ci_builds" SET "runner_id" = 380987, "status" = 'running', "started_at" = '2020-10-29 21:0C "updated_at" = '2020-10-29 21:00:54.568589', "lock_version" = 2 WHERE "ci_builds"."id" = 8201577 */application:web,correlation_id:4ze9HF2IXC9/*
6507699644791286491	SELECT SUM((("project_statistics"."repository_size" + "project_statistics"."lfs_objects_size") - "projec INNER JOIN routes rs ON rs.source_id = projects.id AND rs.source_type = 'Project' INNER JOIN "proj "project_statistics"."project_id" = "projects"."id" WHERE (rs.path LIKE 'gitlab-org/%') AND ("project_s "project_statistics"."lfs_objects_size") > "projects"."repository_size_limit" AND "projects"."repository_s */application:web,controller:merge_requests,action:index,correlation_id:HlfxW7Ir8b1/*

Performance degradation analysis





Jose Finotto @Finotto · 1 day ago

Owner





We had the following top 10 statements **by total calls** in execution during this peak:

Query:

```
topk(10,
  sum by (queryid) (
    rate(pg_stat_statements_calls_total{env="gprd", monitor="db", type="patroni",instance="patroni-06-db-gprd
  )
)
```

In this analysis, we are considering a 15 minutes interval.

https://thanos-query.ops.gitlab.net/graph?g0.range_input=15m&g0.end_input=2021-01-12%2016%3A15&g0.step_input=10&g0.moment_input=2021-01-08%2014%3A15%3A00&g0.max_source_resolution=0s&g0.expr=topk(10%2C%20%0A%20%20sum%20by%20(queryid)%20(%0A%20%20%20%20rate(pg_stat_statements_calls%7Benv%3D%22gprd%22%2C%20monitor%3D%22db%22%2C%20type%3D%22patroni%22%2Cinstance%3D%22patroni-06-db-gprd.c.gitlab-production.internal%3A9187%22%7D%5B1m%5D)%0A%20%20)%0A)&g0.tab=0

Edited by Jose Finotto 1 day ago

Performance degradation analysis





Jose Finotto @Finotto · 1 day ago

Owner   

The outputs are:

Enable query history

```
topk(10,
 sum by (queryid) (
  rate(pg_stat_statements_calls{env="gprd", monitor="db", type="patroni",instance="patroni-06-db-gprd.c.gitlab-production.internal:9187"}[1m])
 )
)
```

Load time: 283ms
Resolution: 10s
Total time series: 17

Execute | - insert metric at cursor - | deduplication | partial response

Graph | Console

− 15m + | ◀◀ 2021-01-12 16:15 ▶▶ | 10 | stacked | Only raw data

[redacted]

Performance degradation analysis



QueryId	Query
833913155023572892	SELECT $1
73367110635711796	SELECT "projects".* FROM "projects" WHERE "projects"."id" = $1 LIMIT $2 */application:web,controller:issues,action:index,correlation_id:tt4UclFKFU9/*
6769309683899657633	SELECT "routes".* FROM "routes" WHERE "routes"."source_id" = $1 AND "routes"."source_type" = $2 LIMIT $3 */application:web,controller:issues,action:index,correlation_id:tt4UclFKFU9/*
6974950735891200787	SELECT "namespaces".* FROM "namespaces" WHERE "namespaces"."id" = $1 LIMIT $2 */application:web,correlation_id:e7d284e6-07ff-4c0e-ae4a-e6880d46b20a/*
6749620766035719574	SELECT "taggings".* FROM "taggings" WHERE "taggings"."taggable_id" = $1 AND "taggings"."taggable_type" = $2 */application:web,controller:projects,action:show,correlation_id:ZiDjjveMIXa/*
6504150523421693673	SELECT "tags".* FROM "tags" INNER JOIN "taggings" ON "tags"."id" = "taggings"."tag_id" WHERE "taggings"."taggable_id" = $1 AND "taggings"."taggable_type" = $2 AND (taggings.context = $3 AND taggings.tagger_id IS NULL) */application:web,correlation_id:dnT2GXhKuX2/*
-2372450153195223637	SELECT $1 AS one FROM ((SELECT "ci_runners".* FROM "ci_runners" INNER JOIN "ci_runner_projects" ON "ci_runner_projects"."runner_id" = "ci_runners"."id" WHERE "ci_runner_projects"."project_id" = $2) UNION ALL (SELECT "ci_runners".* FROM "ci_runners" INNER JOIN "ci_runner_namespaces" ON "ci_runner_namespaces"."runner_id" = "ci_runners"."id" INNER JOIN "namespaces" ON "namespaces"."id" = "ci_runner_namespaces"."namespace_id" AND "namespaces"."type" = $3 WHERE "namespaces"."id" IN (WITH RECURSIVE "base_and_ancestors" AS ((SELECT "namespaces".* FROM "namespaces" INNER JOIN "projects" ON "projects"."namespace_id" = "namespaces"."id" WHERE "namespaces"."type" = $4 AND "projects"."id" = $5) UNION (SELECT "namespaces".* FROM "namespaces", "base_and_ancestors"

Postgres-checkup

Nikolay and his team develop postgres-checkup (https://gitlab.com/postgres-ai/postgres-checkup) -- a tool for automated health-checks of Postgres databases, that contains:



- 28 reports, checking various aspects of Postgres production database health and performing detailed SQL workload analysis.
- Reports contain 3 detailed parts: observations, conclusions, and recommendations.
- Very lightweight checks, unobtrusive activities working well under heavy load, in large databases. Does not require any setup on the servers.
- Multi-node analysis: the master is checked together with its replicas.

postgres-checkup

- Weekly tech audit reports that augment the existing monitoring (prometheus, postgres_exporter, grafana, thanos):
    - track Postgres and components versions
    - track settings and setting deviations
    - bloat control (tables, indexes)
    - index health (invalid, unused, redundant, etc)
    - deep query analysis
    - object sizes
    - int4 PKs
    - ... and more



H002 Unused Indexes

Observations

Data collected: 2021-01-11 13:40:38 +0000 UTC
Current database: gitlabhq_production

Stats reset: 6 mons 27 days 14:26:00 ago (2020-06-13 23:13:01 +0000 UTC)

Never Used Indexes

The list is limited to 50 items. Total: 178.

#	Table	Index	10.220.16.106 usage	10.220.16.101 usage	10.220.16.102 usage	10.220.16.103 usage	10.220.16.104 usage	10.220.16.105 usage	10.220.16.107 usage	10.220.16.108 usage	▼ Index size	Table size
	=====TOTAL=====										165.66 GiB	7.27 TiB
1	ci_builds	index_ci_builds_on_protected	0	0	0	0	0	0	0	0	45.55 GiB	0.92 TiB
2	ci_builds	index_ci_builds_on_user_id_and_created_at_and_type_eq_ci_build	0							0	30.01 GiB	0.92 TiB
3	ci_builds	index_ci_builds_on_queued_at	0	0	0	0	0	0	0	0	22.23 GiB	0.92 TiB
4	merge_request_diffs	index_merge_request_diffs_on_external_diff_store	0	0	0	0	0	0	0	0	8.41 GiB	27.54 GiB
5	projects	index_projects_on_runners_token	0	0	0	0	0	0	0	0	3.90 GiB	4.77 GiB
6	projects	index_projects_on_mirror_last_successful_update_at	0	0	0	0	0	0	0	0	3.90 GiB	4.77 GiB
7	projects	index_projects_on_last_repository_check_failed	0	0	0	0	0	0	0	0	3.89 GiB	4.77 GiB
8	projects	index_projects_on_pending_delete	0	0	0	0	0	0	0	0	3.89 GiB	4.77 GiB
9	users	index_users_on_accepted_term_id	0	0	0	0	0	0	0	0	2.07 GiB	3.55 GiB
10	ci_runners	index_ci_runners_on_is_shared	0	0	0	0	0	0	0	0	2.03 GiB	337.54 MiB
11	merge_request_metrics	index_mr_metrics_on_target_project_id_merged_at_time_to_merge	0	0		0	0	0	0	0	2.00 GiB	6.06 GiB
12	notes	note_mentions_temp_index	0	0	0	0	0	0	0	0	1.75 GiB	299.19 GiB
13	namespaces	index_namespaces_on_shared_and_extra_runners_minutes_limit	0	0	0		0	0	0	0	1.27 GiB	2.56 GiB
14	namespaces	index_namespaces_on_ldap_sync_last_update_at	0	0	0	0	0	0	0	0	1.22 GiB	2.56 GiB

K003 Top-50 Queries by total_time

Observations

Data collected: 2021-01-11 13:40:41 +0000 UTC
Current database: gitlabhq_production

Master (`10.220.16.106`)

Start: 2021-01-11T13:05:57.091968+00:00
End: 2021-01-11T13:39:07.728772+00:00
Period seconds: 1990.6368
Period age: 00:33:10.636804

Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

The list is limited to 50 items.

# (query id)	Query	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	shar
1 (-6386890822646777000)	SELECT "users".* FROM "users" INNER JOIN "project_authorizations" ON "users"."id" = "project_authorizations"."user_id" WHERE "project_authorizations"."project_id" = $1 /*application:web,correlation_id:Lmz5Aaf8Vpa*/ Full query	72,767 36.55/sec 1.00/call 0.16%	1,140,899.14 ms 573.133 ms/sec 15.679 ms/call 15.52%	7,371,979 3.71K/sec 101.31/call 4.57%	33,889,906 blks 17.03K blks/sec 465.73 blks/call 2.86%	816,616 blks 410.23 blks/sec 11.22 blks/call 9.36%	4,870 blks 2.45 blks/sec 0.07 blks/call 0.20%	148 b 0.07 0.00 3.12
2 (-7232084447659837000)	WITH RECURSIVE "namespaces_cte" AS ((SELECT "namespaces"."id", "members"."access_level" FROM "namespaces" INNER JOIN "members" ON "namespaces"."id" = "members"."source_id" WHERE "members"."type" = $1 AND "members"."source_type" = $2 AND "namespaces"."type" = $3 AND "members"."user_id" = $4 AND "members"."requested_at" IS NULL AND (access_level >= $5)) UNION (SELECT "namespaces"."id", LEAST("members"."access_level", "group_group_links"."group_access") AS access_level FROM "namespaces" INNER JOIN "group_group_links" ON "group_group_links"."shared_group_id" = "namespaces"."id" INNER JOIN "members" ON "group_group_links"."shared_with_group_id" = "members"."source_id" AND "members"."source_type" = $6 AND "members"."requested_at" IS NULL AND "members"."user_id" = $7 AND "members"."access_level" > $8 WHERE "namespaces"."type" = $9) UNION (SELECT "namespaces"."id", GREATEST("members"."access_level"	41,162 20.68/sec 1.00/call 0.09%	995,280.30 ms 499.981 ms/sec 24.180 ms/call 13.54%	89,469,596 44.95K/sec 2.18K/call 55.44%	504,881,421 blks 253.63K blks/sec 12.27K blks/call 42.55%	33,166 blks 16.66 blks/sec 0.81 blks/call 0.38%	954 blks 0.48 blks/sec 0.02 blks/call 0.04%	1 blk 0.00 0.00 0.02

K002 Workload Type ("The First Word" Analysis)

Observations

Data collected: 2021-01-11 13:40:41 +0000 UTC
Current database: gitlabhq_production

Master (`10.220.16.106`)

Start: 2021-01-11T13:05:57.091968+00:00
End: 2021-01-11T13:39:07.728772+00:00
Period seconds: 1990.6368
Period age: 00:33:10.636804

Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

#	Workload type	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	shared_blks_written	blk_read_time	blk_write_time	kcach
1	select	41,827,896 21.02K/sec 1.00/call 94.08%	5,019,032.35 ms 2521.320 ms/sec 0.120 ms/call 68.28%	68,804,876 34.57K/sec 1.64/call 42.63%	604,189,040 blks 303.52K blks/sec 14.44 blks/call 50.92%	6,911,927 blks 3.48K blks/sec 0.17 blks/call 79.21%	95,009 blks 47.73 blks/sec 0.00 blks/call 3.81%	3,913 blks 1.97 blks/sec 0.00 blks/call 82.55%	1,373,499.10 ms 689.980 ms/sec 0.033 ms/call 73.22%	92.86 ms 0.047 ms/sec 0.000 ms/call 81.72%	0.00 b 0.00 b 0.00 b 0.00%
2	with	752,897 378.22/sec 1.00/call 1.69%	1,066,397.37 ms 535.707 ms/sec 1.416 ms/call 14.51%	90,927,447 45.68K/sec 120.77/call 56.34%	512,703,313 blks 257.56K blks/sec 680.97 blks/call 43.21%	33,166 blks 16.66 blks/sec 0.04 blks/call 0.38%	954 blks 0.48 blks/sec 0.00 blks/call 0.04%	1 blks 0.00 blks/sec 0.00 blks/call 0.02%	6,402.21 ms 3.216 ms/sec 0.009 ms/call 0.34%	0.04 ms 0.000 ms/sec 0.000 ms/call 0.04%	0.00 b 0.00 b 0.00 b 0.00%
3	update	999,462 502.08/sec 1.00/call 2.25%	755,406.99 ms 379.480 ms/sec 0.756 ms/call 10.28%	741,537 372.51/sec 0.74/call 0.46%	48,497,397 blks 24.37K blks/sec 48.52 blks/call 4.09%	1,211,149 blks 608.42 blks/sec 1.21 blks/call 13.88%	1,681,451 blks 844.68 blks/sec 1.68 blks/call 67.36%	533 blks 0.27 blks/sec 0.00 blks/call 11.24%	213,446.15 ms 107.225 ms/sec 0.214 ms/call 11.38%	13.47 ms 0.007 ms/sec 0.000 ms/call 11.85%	0.00 b 0.00 b 0.00 b 0.00%
4	insert	837,581 420.76/sec 1.00/call 1.88%	502,066.21 ms 252.214 ms/sec 0.599 ms/call 6.83%	873,121 438.61/sec 1.04/call 0.54%	21,046,160 blks 10.58K blks/sec 25.13 blks/call 1.77%	561,911 blks 282.28 blks/sec 0.67 blks/call 6.44%	711,983 blks 357.67 blks/sec 0.85 blks/call 28.52%	289 blks 0.15 blks/sec 0.00 blks/call 6.10%	282,057.24 ms 141.692 ms/sec 0.337 ms/call 15.04%	7.18 ms 0.004 ms/sec 0.000 ms/call 6.32%	0.00 b 0.00 b 0.00 b 0.00%
5	select ... for [no key] update	40,689 20.44/sec 1.00/call 0.09%	7,361.69 ms 3.698 ms/sec 0.181 ms/call 0.10%	40,689 20.44/sec 1.00/call 0.03%	207,484 blks 104.23 blks/sec 5.10 blks/call 0.02%	7,997 blks 4.02 blks/sec 0.20 blks/call 0.09%	6,751 blks 3.39 blks/sec 0.17 blks/call 0.27%	4 blks 0.00 blks/sec 0.00 blks/call 0.08%	361.00 ms 0.181 ms/sec 0.009 ms/call 0.02%	0.08 ms 0.000 ms/sec 0.000 ms/call 0.07%	0.00 b 0.00 b 0.00 b 0.00%

Replica servers:

Replica (`10.220.16.101`)

Start: 2021-01-11T13:05:51.048781+00:00
End: 2021-01-11T13:36:10.229216+00:00
Period seconds: 1819.18044
Period age: 00:30:19.180435

#	Workload type	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	shared_blks_written	blk_read_time	blk_write_time	kcac
1	select	20,300,433 11.16K/sec	9,206,677.68 ms 5060.893 ms/sec	29,084,119 15.99K/sec	4,829,058,098 blks 2.66M blks/sec	12,903,398 blks 7.10K blks/sec	0 blks 0.00 blks/sec	401,782 blks 220.86 blks/sec	866,803.16 ms 476.480 ms/sec	10,641.69 ms 5.850 ms/sec	0.00 0.00

K001 Globally Aggregated Query Metrics

Observations

Data collected: 2021-01-11 13:40:41 +0000 UTC
Current database: gitlabhq_production

Master (`10.220.16.106`)

Start: 2021-01-11T13:05:57.091968+00:00
End: 2021-01-11T13:39:07.728772+00:00
Period seconds: 1990.6368
Period age: 00:33:10.636804

Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

Calls	Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	shared_blks_written	blk_read_time	b
44,458,525 22.34K/sec 1.00/call 100.00%	7,350,264.61 ms 3692.419 ms/sec 0.165 ms/call 100.00%	161,387,670 81.08K/sec 3.63/call 100.00%	1,186,643,394 blks 596.12K blks/sec 26.69 blks/call 100.00%	8,726,150 blks 4.39K blks/sec 0.20 blks/call 100.00%	2,496,148 blks 1.26K blks/sec 0.06 blks/call 100.00%	4,740 blks 2.38 blks/sec 0.00 blks/call 100.00%	1,875,765.69 ms 942.294 ms/sec 0.042 ms/call 100.00%	1 0 0 1

Replica servers:

Replica (`10.220.16.101`)

Start: 2021-01-11T13:05:51.048781+00:00
End: 2021-01-11T13:36:10.229216+00:00
Period seconds: 1819.18044
Period age: 00:30:19.180435

Calls	Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	shared_blks_written	blk_read_time	blk
21,518,698 11.83K/sec 1.00/call 100.00%	9,393,835.91 ms 5163.774 ms/sec 0.437 ms/call 100.00%	31,311,433 17.22K/sec 1.46/call 100.00%	4,887,880,546 blks 2.69M blks/sec 227.15 blks/call 100.00%	12,921,880 blks 7.11K blks/sec 0.60 blks/call 100.00%	0 blks 0.00 blks/sec 0.00 blks/call 0.00%	402,350 blks 221.17 blks/sec 0.02 blks/call 100.00%	868,082.59 ms 477.183 ms/sec 0.040 ms/call 100.00%	10, 5.8 0.0 10

Replica (`10.220.16.102`)

Start: 2021-01-11T13:05:52.314852+00:00
End: 2021-01-11T13:36:39.061152+00:00
Period seconds: 1846.7463
Period age: 00:30:46.7463

Calls	Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	shared_blks_written	blk_read_time	blk
23,905,903 12.95K/sec 1.00/call	9,934,511.91 ms 5379.468 ms/sec 0.416 ms/call	32,988,938 17.87K/sec 1.38/call	5,015,880,782 blks 2.72M blks/sec 209.82 blks/call	13,968,380 blks 7.57K blks/sec 0.58 blks/call	0 blks 0.00 blks/sec 0.00 blks/call	514,997 blks 278.87 blks/sec 0.02 blks/call	758,241.41 ms 410.582 ms/sec 0.032 ms/call	11, 6.1 0.0

Postgres.ai

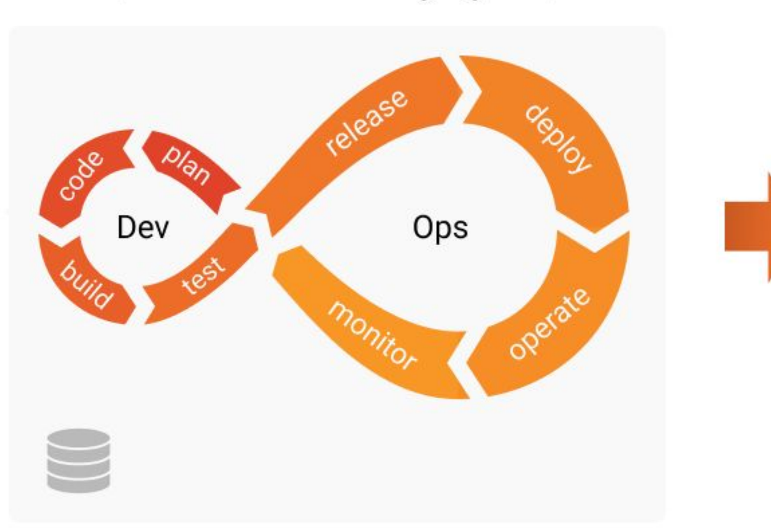
– clone DB of any size in a few seconds in bring them in any point of the DevOps lifecycle

- automated (in CI) testing of DB migrations
- guess-free SQL optimization
- instant deployment of full-size staging apps

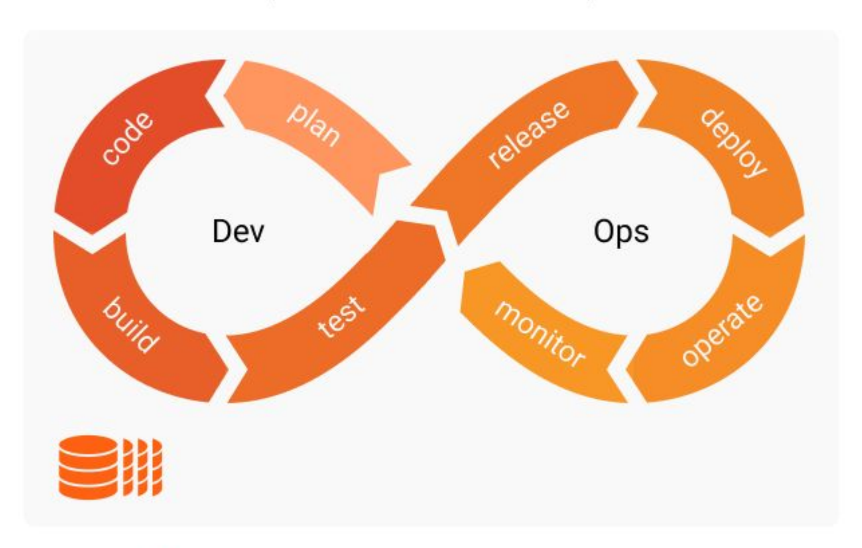


Non-production environment weaknesses are reasons of multiple development problems

Development bottlenecks
(with standard staging DB)



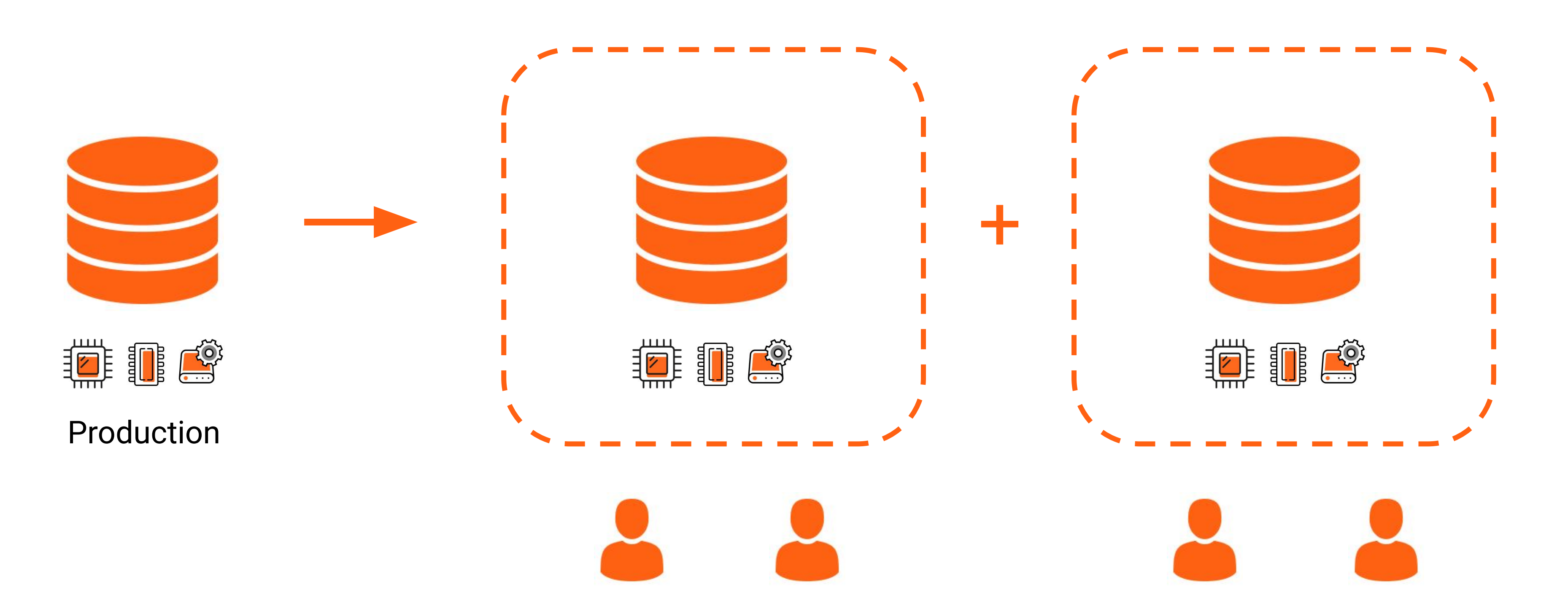
Frictionless development
(with Database Lab)



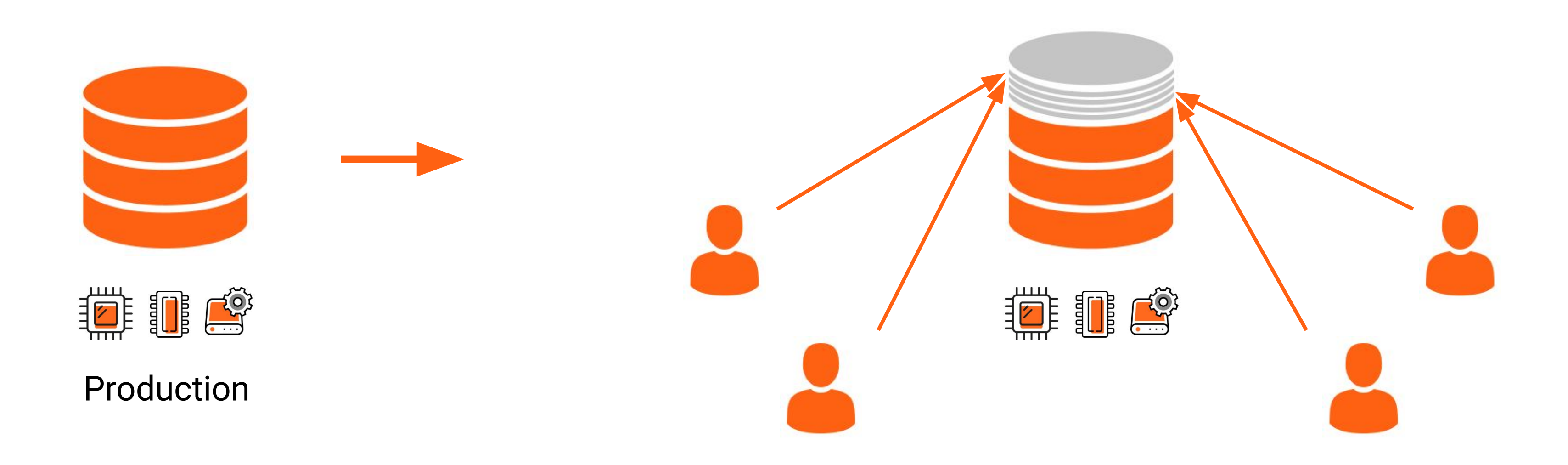
- ✗ Bugs: difficult to reproduce, easy to miss
- ✗ Not 100% of changes are well-verified
- ✗ SQL optimization is hard
- ✗ Each non-prod big DB costs a lot
- ✗ Non-prod DB refresh takes hours, days, weeks

- ✓ Bugs: easy to reproduce, and fix early
- ✓ 100% of changes are well-verified
- ✓ SQL optimization can be done by anyone
- ✓ Non-prod DB refresh takes seconds
- ✓ Extra non-prod DBs doesn't cost a penny

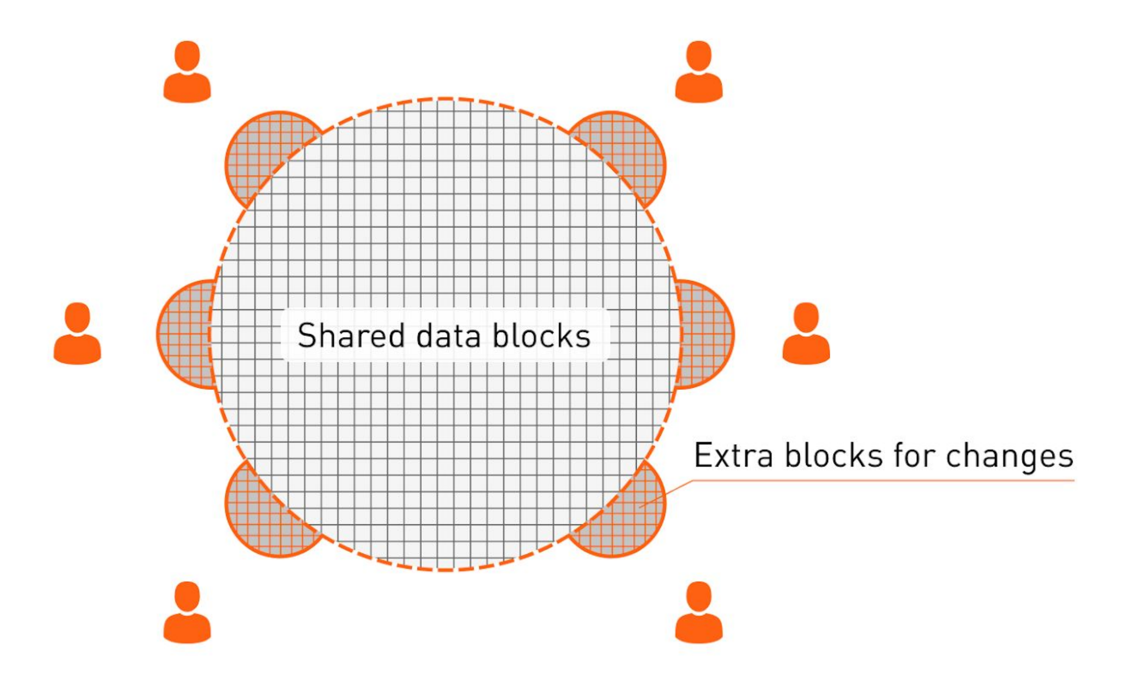
Database experiments – traditional approach



Database experiments on thin clones



Thin clones – copy-on-write



Thick copy of production (any size)

Thin clone (size starts from 1 MB, depends on changes)

Database experiments on thin clones – yes and no

Yes

- Check execution plan – Joe bot
  - EXPLAIN w/o execution
  - EXPLAIN (ANALYZE, BUFFERS)
    - (timing is different; structure and buffer numbers – the same)
- Check DDL
  - index ideas (Joe bot)
  - auto-check DB migrations
- Heavy, long queries: analytics, dump/restore
  - No penalties! (think hot_standby_feedback, locks, CPU)

Joe

No

- Load testing
- Regular HA/DR goals
  - backups
    - (but useful to check WAL stream, recover records by mistake)
  - hot standby
    - (but useful to offload very long-running SELECTs)

Database Lab – Open-core model



Database Lab Engine

Open-source (AGPLv3)

- Thin cloning
- Automated provisioning and data refresh
- Data transformation, anonymization
- Supports managed Postgres (AWS RDS, etc.)

https://gitlab.com/postgres-ai/database-lab

Platform

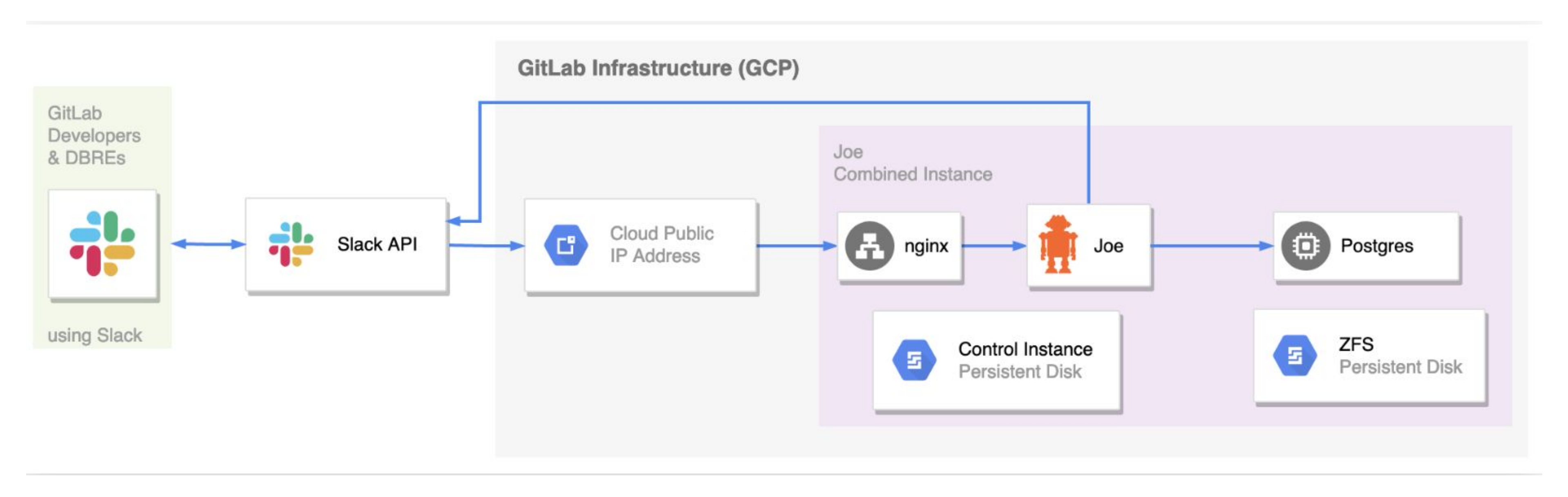
SaaS (pricing model: $ per TiB)

- Web console (GUI)
- Access control, audit
- History, visualization
- Support

https://postgres.ai/

– follow the links and start using it for your databases

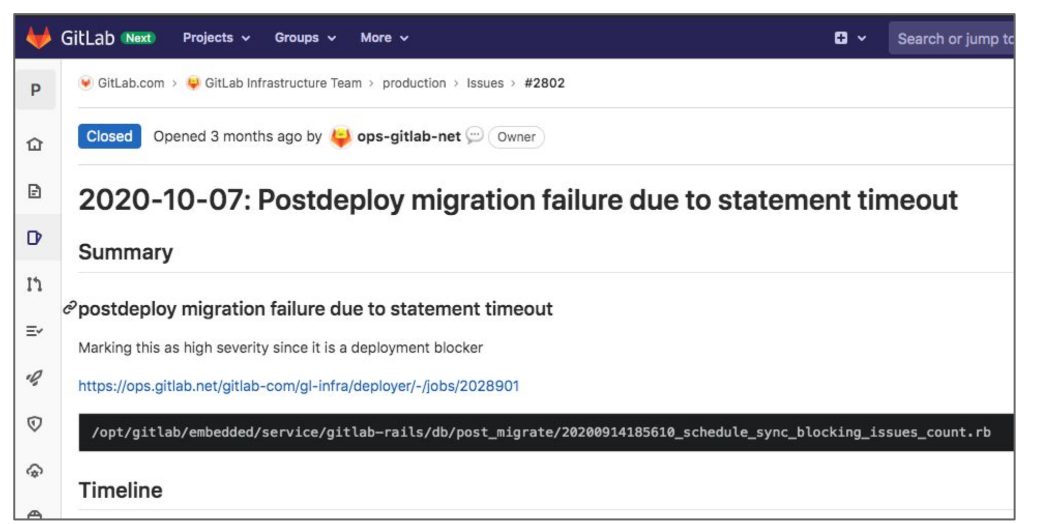
SQL optimization using Database Lab and Joe bot



Automated checks of database migrations (DDL) using full-size thin clones provided by Database Lab

Before Database Lab:

- Developers test DDL on tiny databases, using only synthetic data, not seeing real behavior
- Before each release, DDL is tested on staging – a reduced/old/modified data set (~5-10% of real size)
- Manual code review. Very rarely the change is tested on a production clone



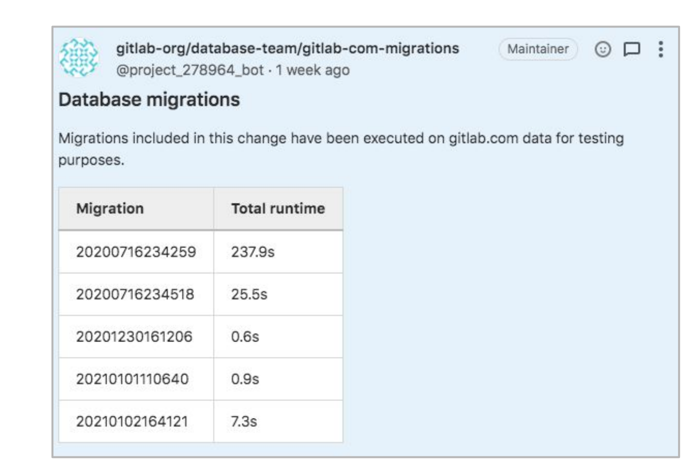
Issues with deploying DB migrations were not uncommon

An example:
https://gitlab.com/gitlab-com/gl-infra/production/-/issues/2802

Automated checks of database migrations (DDL) using full-size thin clones provided by Database Lab

With Database Lab:

- Separate project
  - security: limited access, firewall
  - isolation: reduced codebase and no extra components
  - connected to DLE API, able to use `dblab clone`
- On any CI build in the main project ("gitlab") has DDL, then:
  - a CI build in this special project is triggered
  - DDL is auto-verified on a fresh clone (lag <6h) provided by DLE
  - detailed artifacts are available to the Database Team and Infrastructure
    - Output
    - pg_stat_***
    - production timing estimates
    - Postgres logs
    - pgsa sampling
    - summary
  - summary is automatically posted as an MR comment

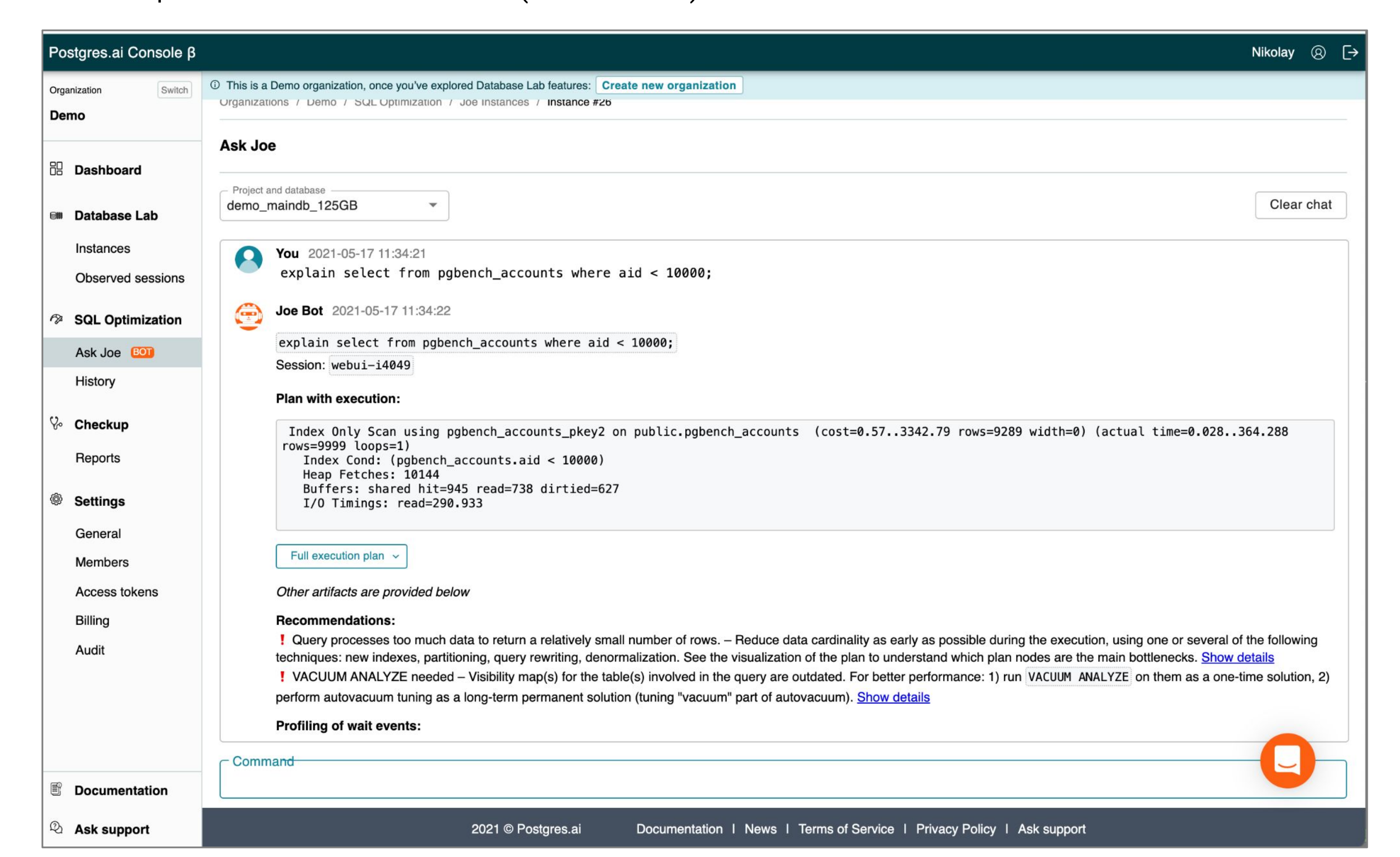

gitlab-org/database-team/gitlab-com-migrations
@project_278964_bot · 1 week ago
Maintainer

Database migrations

Migrations included in this change have been executed on gitlab.com data for testing purposes.

Migration	Total runtime
20200716234259	237.9s
20200716234518	25.5s
20201230161206	0.6s
20210101110640	0.9s
20210102164121	7.3s

SQL Optimization chatbot (“Joe bot”)



SQL Optimization chatbot (“Joe bot”) – summary for a single query

Profiling of wait events:

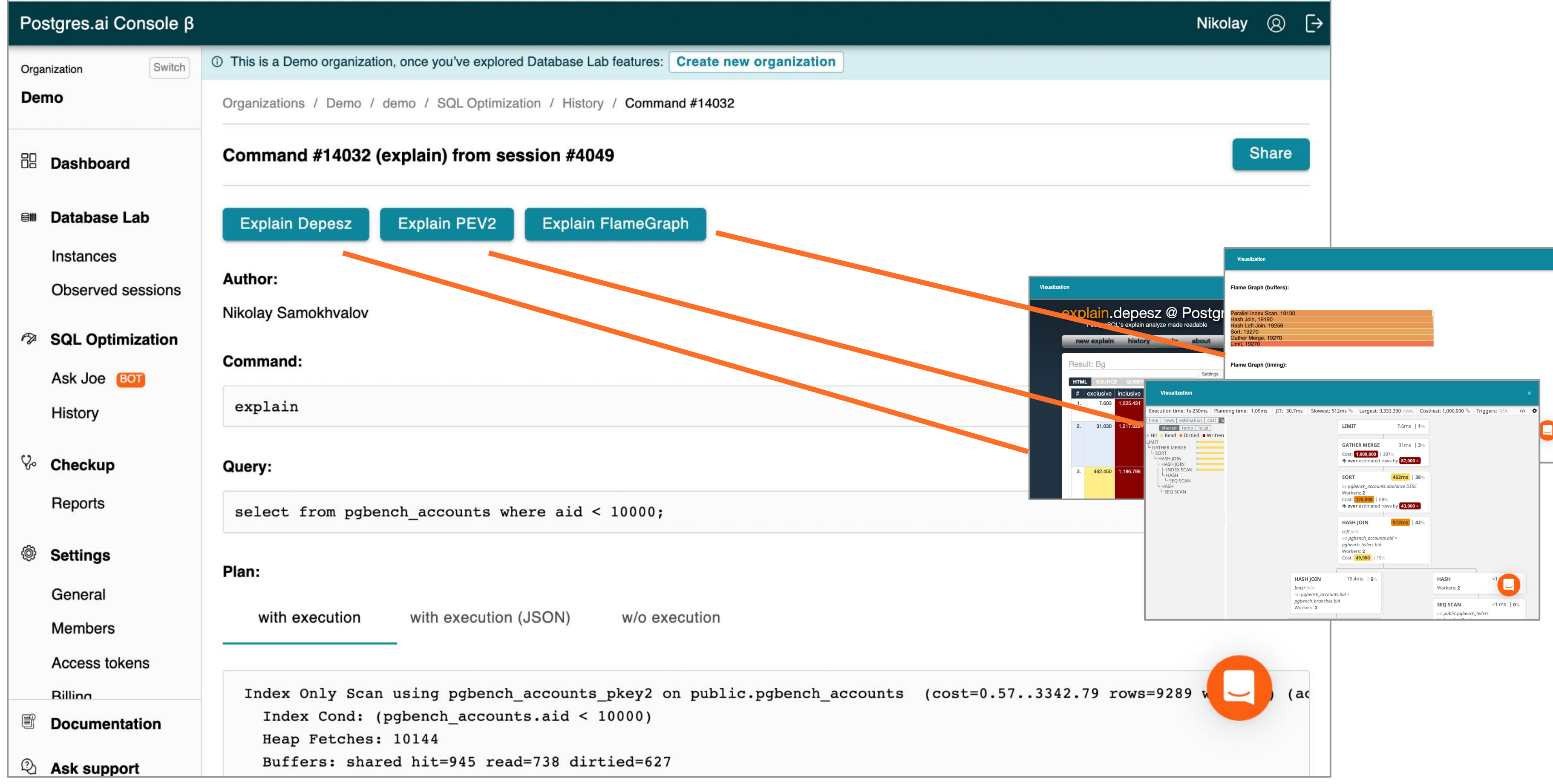
```
% time      seconds wait_event
------ ------------ ------------------------------
70.82      0.256244 IO.DataFileRead
14.63      0.052938 Running
14.55      0.052630 IO.SLRURead
------ ------------ ------------------------------
100.00     0.361812
```

Summary:

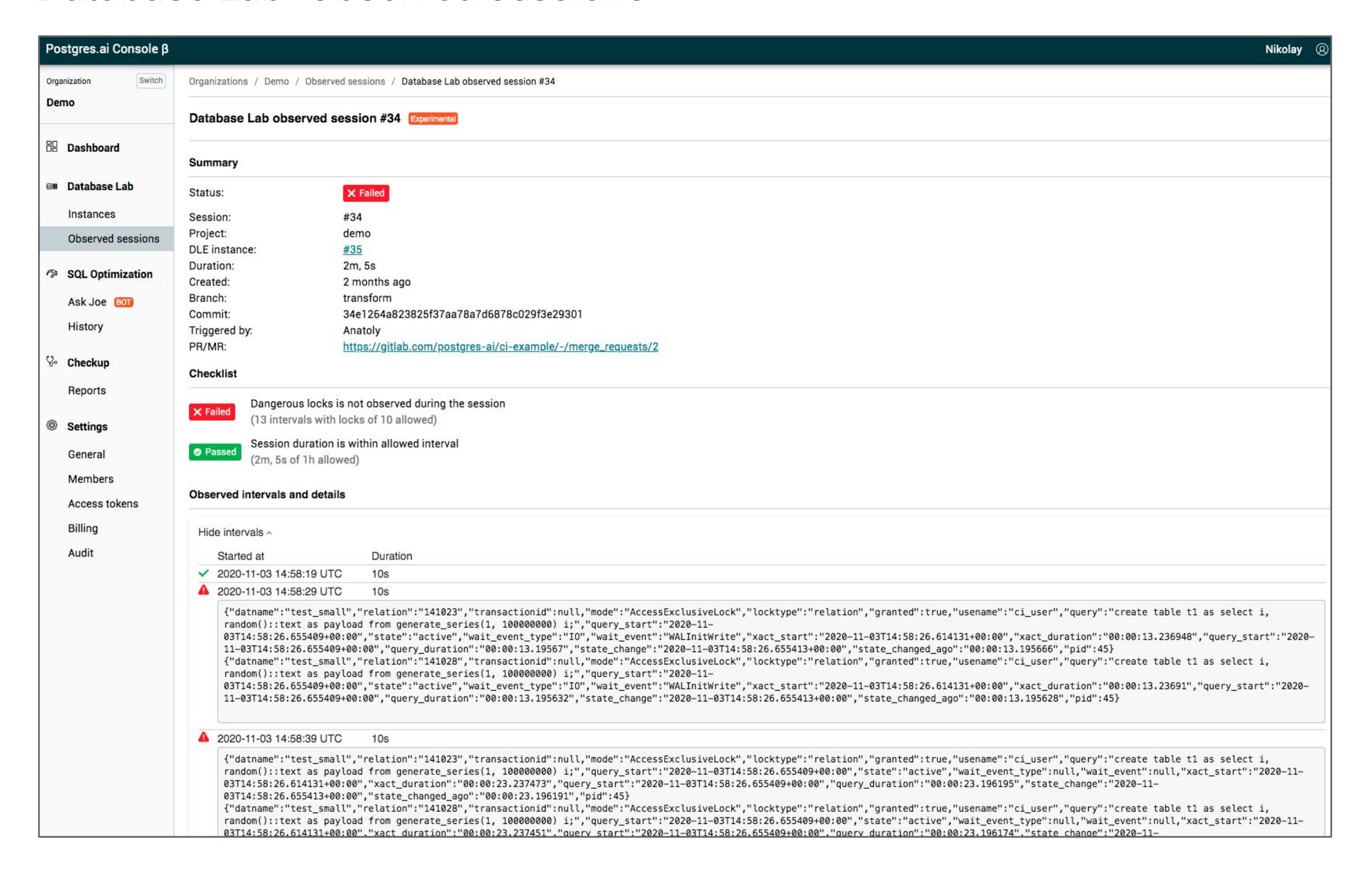
```
Time: 366.157 ms
  - planning: 1.104 ms
  - execution: 365.053 ms (estimated* for prod: 0.053...0.310 s)
    - I/O read: 290.933 ms
    - I/O write: N/A

Shared buffers:
  - hits: 945 (~7.40 MiB) from the buffer pool
  - reads: 738 (~5.80 MiB) from the OS file cache, including disk I/O
  - dirtied: 627 (~4.90 MiB)
  - writes: 0
```

SQL Optimization chatbot ("Joe bot") – History & Visualization



Database Lab “Observed sessions”



Summary

- PostgreSQL database health check is automated
- All engineers now can do the following without delays:
    - get `EXPLAIN (ANALYZE, BUFFERS)` for any query for full-size DBs (not being blocked and not blocking others)
    - get insights of how DDL behaves before submitting MR for DB migration review
    - learn SQL by example (using full-size databases!)
- Database team has
    - Way to conduct various database experiments without need to provision new nodes and/or wait for long data refresh
    - DB migration reviews are pre-checked automatically in 100% of cases, with prediction of what would happen during production deployment

Contribute



Thank you. Please feel free to follow up!

Jose Finotto

jfinotto@gitlab.com

LinkedIn: linkedin.com/in/jose-c-bb4a2178/

Nikolay Samokhvalov

nik@postgres.ai

Twitter: @samokhvalov

LinkedIn: linkedin.com/in/samokhvalov/